CoRRAM Version 2: User Guide

Alfred Maußner
University of Augsburg
27 November 2012

Contents

1 Introduction **2**

2 Installation **2**

3 Structure of the Models **4**
3.1 Variables 4
3.2 Equations 5
3.3 Example 5

4 Solutions **6**
4.1 Steady State 7
4.2 Linearization 8
4.3 Structure of Solutions 13
4.4 Additional Variables 13
4.5 Difference Stationary Growth 14
4.6 Error Messages 15

5 Example Program **15**

6 Program Options **21**

7 Summary of Global Control Variables **22**

8 Inside the Black Box **24**
8.1 Linear Solutions 25
8.2 Quadratic Part of the Solution 25
8.3 Simulation 26

9 References **26**

1 Introduction

The Gauss program CoRRAM provides a framework for computing recursive representative agent models along the lines of Heer and Maußner (2009a), Chapter 2 and Heer and Maußner (2009b). Its aim is to allow less experienced users of Gauss to solve and simulate their model using either linear or quadratic approximations of the model's equilibrium conditions.

I assume that you are familiar with dynamic stochastic general equilibrium (DSGE) models and, in particular, with perturbation methods that obtain approximate solutions of these models. I also assume that you know how to start the Gauss software under the Windows operating system and that you have a basic knowledge of the Gauss syntax.

In this document I use the typewriter font to print Gauss commands and file names, typeset vectors in boldface, and denote the number of elements in a vector $\mathbf{x}$ by $n(x)$.

CoRRAM Version 2 differs from the previous version in three respects:

1. It dispenses with `#include` commands. The program's functionality is provided by Gauss procedures.

2. It includes several additional features. In particular, if you run Gauss version 12.1 or higher, you can use the new graphic capabilities of Gauss.

3. I cleaned up the code in the core, which makes the program more efficient.

2 Installation

The code of CoRRAMis stored in various files. The files with the `.src` and the `.dec` extension contain source code that you should not change. Gauss will look for these files in its `src` subdirectory. For instance, if `c:\gauss` is the directory where Gauss is installed then copy the files to `c:\gauss\src`.

The file `Tools.dll` must be placed in the Gauss `dlib` subdirectory (for instance in `c:\Gauss\dlib`). This dynamic link library contains routines written in Fortran. In order to use this library you also have to install the file `DFORRT.dll` in the `system32` subdirectory of your Windows 32 bit operating system (i.e. in `c:\windows\system32`) or in the `SysWOW64` subdirectory of your Windows 64 bit system. If you do not want to use these routines, you can set the flag `_GaussOnly=1`. Yet, in this case, you will not be able to use routines that require the generalized Schur factorization.

There are two versions of the CoRRAM Gauss library file. If you run Gauss version 12.1 or higher copy the file `CoRRAM-1.lcg` to the Gauss `lib` directory, e.g., in

c:\gauss\lib and rename the file to CoRRAM.lcg. If you run Gauss version 8-12, copy the file CoRRAM-2.lcg to the Gauss lib directory and rename it CoRRAM.lcg.

There are also two versions of a file that defines certain Gauss structures. For Gauss versions 12.1 or higher copy CoRRAM-1.sdf to the Gauss scr directory and rename it to CoRRAM.sdf. For Gauss versions 8-12 copy the file CoRRAM-2.sdf to src and rename it to CoRRAM.sdf.

The files CoRRAM_Test_1.g, CoRRAM_Test_2.g, and CoRRAM_Test_3.g illustrate the use of the program with the benchmark business cycle model as it is described in Heer and Maußner (2009a), Section 2.6.1. You should run these files to check your installation. Table 2.1 summarizes the file installation.

Table 2.1
File Installation

File	Destination	Remark
CoRRAM_1.src	[GaussDir]\src\CoRRAM_1.src	
CoRRAM_2.src	[GaussDir]\src\CoRRAM_2.src	
CoRRAM_3.src	[GaussDir]\src\CoRRAM_3.src	
CoRRAM_4.src	[GaussDir]\src\CoRRAM_4.src	
CoRRAM_5.src	[GaussDir]\src\CoRRAM_5.src	
CoRRAM.dec	[GaussDir]\src\CoRRAM.dec	
Plot.src	[GaussDir]\src\Plot.src	
Plot-b.src	[GaussDir]\src\Plot_b.src	
CoRRAM-1.sdf	[GaussDir]\src\CoRRAM.sdf	for Gauss version 12.1 and higher
CoRRAM-2.sdf	[GaussDir]\src\CoRRAM.sdf	for Gauss version 8-12
CoRRAM-1.lcg	[GaussDir]\lib\CoRRAM.lcg	for Gauss version 12.1 and higher
CoRRAM-2.lcg	[GaussDir]\lib\CoRRAM.lcg	for Gauss version 8-12
DFORRT.dll	[WinDir]\system32\DFORRT.dll	for Windows 32 bit
DFORRT.dll	[WinDir]\SysWOW64\DFORRT.dll	for Windows 64 bit
CorRRAM_Test_1.g	the users working directory	
CorRRAM_Test_2.g	the users working directory	
CorRRAM_Test_3.g	the users working directory	

Notes: [GaussDir] refers to the directory where you have installed Gauss. [WinDir] refers to the directory where your Windows operation system is installed.

3 Structure of the Models

In this section I describe the canonical form of the models that the CoRRAM-program is able to solve. Later sections will consider different ways to input your model.

3.1 Variables

Basically, DSGE models have three different kinds of variables:

1. Purely exogenous variables, the model's shocks,
2. state variables, i.e., endogenous variables, whose values at the beginning of time t are predetermined (as, e.g., the stock of capital or the stock of nominal bonds),
3. endogenous variables, whose values are determined within period t.

I stack the shocks of period t in the column vector $\mathbf{z}_t = \left[z_{1t}, z_{2t}, \ldots, z_{n(z)t}\right]'$. Analogously, the column vectors $\mathbf{x}_t = \left[x_{1t}, x_{2t}, \ldots, x_{n(x)t}\right]'$ and $\mathbf{y}_t = \left[y_{1t}, y_{2t}, \ldots, y_{n(y)t}\right]'$ collect the $n(x)$ state variables and the $n(y)$ remaining endogenous variables, respectively.

With respect to the model's shocks, the programs assume that either their absolute or their relative deviations from their unconditional means follow the first-order autoregressive vector process

$$\mathbf{z}_t = \Pi \mathbf{z}_{t-1} + \boldsymbol{\epsilon}_t, \tag{3.1a}$$

$$\boldsymbol{\epsilon}_t = \sigma \Omega \boldsymbol{\nu}_t, \quad \sigma \geq 0, \tag{3.1b}$$

$$\boldsymbol{\nu}_t \sim N(\mathbf{0}_{n(z)\times 1}, I_{n(z)}). \tag{3.1c}$$

In order for this process to be stationary, the eigenvalues of the matrix Π must all be located within the unit circle. Note that the covariance matrix Σ of the vector $\boldsymbol{\epsilon}_t$ is

$$\Sigma = \mathbb{E}\left(\boldsymbol{\epsilon}_t \boldsymbol{\epsilon}_t'\right) = \mathbb{E}\left(\sigma \Omega \boldsymbol{\eta}_t \boldsymbol{\eta}_t' \Omega' \sigma\right) = \sigma \Omega I_{nz} \Omega' \sigma = \sigma^2 \Omega \Omega'.$$

Thus, if the positive definite matrix Σ rather then Ω is given and the scaling factor σ is set equal to unity, Ω can be computed from

$$\Omega = C \Lambda^{1/2}$$

where

- $\Lambda^{1/2}$ is the diagonal matrix with the square roots of the eigenvalues of the matrix Σ on the main diagonal, and
- C is the matrix of the normalized eigenvectors of Σ so that $CC' = I_{n(z)}$.

You can use the procedure `GetOmega` (documented in the file `CoRRAM_3.src`) to obtain Ω from Σ.

3.2 Equations

The dynamics of the models must be governed by the set of equations:

$$\mathbb{E}_t g^i \left(\mathbf{x}_t, \mathbf{y}_t, \mathbf{z}_t, \mathbf{x}_{t+1}, \mathbf{y}_{t+1}, \mathbf{z}_{t+1} \right) = 0, \quad i = 1, 2, \ldots, n(x) + n(y), \tag{3.2}$$

where $\mathbb{E}_t$ denotes expectations as of period t. Usually, a subset of $n(u)$ of the equations will only involve date t dated variables. In this case, the dimension of the linearized dynamic model can be reduced. Towards this purpose we partition the vector $\mathbf{y}_t$:

$$\mathbf{y}_t = \begin{bmatrix} \mathbf{u}_t \\ \boldsymbol{\lambda}_t \end{bmatrix}. \tag{3.3}$$

I refer to the $n(u)$ variables in the vector $\mathbf{u}_t$ as control variables (this naming is borrowed from the linear-quadratic control problem) and to the remaining variables in the $n(\lambda)$ vector $\boldsymbol{\lambda}_t$ as costate variables. Technically, the $n(u)$ static equations determine u_{1t} through $u_{n(u)t}$ for given values of the shocks $\mathbf{z}_t$, the state variables $\mathbf{x}_t$ and the costate variables $\boldsymbol{\lambda}_t$. However, while the initial values of the variables in $\mathbf{x}_t$ are truly given, the initial values of the costate variables are chosen by the solution algorithm to satisfy the model's transversality conditions.

3.3 Example

The stationary variables of the benchmark business cycle model from Section 1.5.1 of Heer and Maußner (2009a) are the capital stock per efficiency unit, k_t, output, consumption, and investment per efficiency unit, y_t, c_t, and i_t, respectively, working hours N_t, the rental rate of capital r_t, the real wage per efficiency unit of labor w_t, the level of total factor productivity Z_t, and the scaled Lagrange multiplier of the household's budget constraint λ_t. Thus:

- $\mathbf{x}_t = k_t$,
- $\mathbf{y}_t = [y_t, c_t, i_t, N_t, w_t, r_t, \lambda_t]'$,
- $z_t = \ln Z_t$.

The model's equations are:

$$0 = c_t^{-\eta}(1-N_t)^{\theta(1-\eta)} - \lambda_t, \tag{3.4a}$$

$$0 = \theta c_t^{1-\eta}(1-N_t)^{\theta(1-\eta)-1} - \lambda_t w_t, \tag{3.4b}$$

$$0 = r_t - \alpha\frac{y_t}{k_t}, \tag{3.4c}$$

$$0 = w_t - (1-\alpha)\frac{y_t}{N_t}, \tag{3.4d}$$

$$0 = y_t - Z_t N_t^{1-\alpha} k_t^{\alpha}, \tag{3.4e}$$

$$0 = y_t - i_t - c_t, \tag{3.4f}$$

$$0 = ak_{t+1} - (1-\delta)k_t - i_t, \tag{3.4g}$$

$$0 = \lambda_t - \beta a^{-\eta}\mathbb{E}_t\lambda_{t+1}\left(1-\delta+r_{t+1}\right), \tag{3.4h}$$

where a, α, β, δ, η, and θ are parameters of the model. Equations (3.4a) through (3.4f) are static equations (i.e., they only involve variables with time index t) and, thus, 6 of the 7 variables in the vector $\mathbf{y}_t$ can be determined for given values of k_t, Z_t, and one other variable. For instance, if we use λ_t as costate variable, $\boldsymbol{\lambda}_t \equiv \lambda_t$, the vector $\mathbf{u}_t$ consists of y_t, c_t, i_t, N_t, w_t, and r_t.

Finally, with $\mathbb{E}Z_t = Z \equiv 1$, $z_t := \ln Z_t \simeq ((Z_t)-1)/1$ evolves according to

$$z_{t+1} = \varrho z_t + \sigma^Z \nu_{t+1}, \quad \nu \sim N(0,1)$$

so that $\mathbf{z}_t \equiv z_t$, $\Pi \equiv \varrho$, $\Omega \equiv 1$, and $\sigma \equiv \sigma^Z$ is the standard deviation of the innovation in the AR(1)-process of the technology shock $\ln Z_t$.

4 Solutions

Note that for $\sigma = 0$ the model in (3.1) and (3.2) collapses to a deterministic model, in which all shocks equal their conditional means of zero. Given that the deterministic model is stable, it will approach a stationary solution (or a balanced growth path) in which all variables remain constant. The CoRRAM program computes linear and quadratic approximations of the stochastic model's solution at this point.

There are two ways to compute the quadratic part. If the global control variable `_QA_type` is set to zero, the code is based on Heer and Maußner (2009a), pp. 124-131. If this variable is set to one, the algorithm of Gomme and Klein (2011) is employed. In this case, the global control variable `_sylvester` determines if the respective Sylvester equation is solved by applying the vec operator or by a call to a routine stored in `Tools.dll`. The default settings of `_QA_type` and `_sylvester` are 1.

Both the linear and the quadratic approximations build on the linearized equations (3.2). In order to understand the way you can input your model and to be able to

respond to error messages reported by the program you need a basic understanding of the linearized model.

4.1 Steady State

The stationary solution is obtained from (3.2) in the following steps:

1. Replace $\mathbf{z}_t$ by its unconditional mean $\mathbf{z} = \mathbf{0}$ so that the expectations operator $\mathbb{E}_t$ can be ignored.

2. Assume stationarity, i.e., drop the time index from all variables.

3. Solve

$$g^i\left(\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{x}, \mathbf{y}, \mathbf{z}\right) = 0, \quad i = 1, 2, \ldots, n(x) + n(y),$$

   for $\mathbf{x}$ and $\mathbf{y}$.

There are two ways to compute this solution: a) Use paper and pencil to reduce the system so that it can be solved equation by equation, and b) employ a non-linear equations solver.

Step by Step. Consider example (3.4). In the steady state with $Z \equiv 1$ it reduces to:

$$0 = c^{-\eta}(1-N)^{\theta(1-\eta)} - \lambda, \tag{4.1a}$$
$$0 = \theta c^{1-\eta}(1-N)^{\theta(1-\eta)-1} - \lambda w, \tag{4.1b}$$
$$0 = r - \alpha\frac{y}{k}, \tag{4.1c}$$
$$0 = w - (1-\alpha)\frac{y}{N}, \tag{4.1d}$$
$$0 = y - N^{1-\alpha}k^{\alpha}, \tag{4.1e}$$
$$0 = y - i - c, \tag{4.1f}$$
$$0 = ak - (1-\delta)k - i, \tag{4.1g}$$
$$0 = \lambda - \beta a^{-\eta}\lambda\left(1 - \delta + r\right). \tag{4.1h}$$

Substituting equation (4.1c) into (4.1h) we can solve for the output-capital ratio:

$$\frac{y}{k} = \frac{a^{\eta} - \beta(1-\delta)}{\alpha\beta}. \tag{4.2a}$$

The resource constraint (4.1f) and equation (4.1g) imply

$$\frac{c}{k} = \frac{y}{k} - (a - 1 + \delta). \tag{4.2b}$$

Substituting for λ in (4.1b) from (4.1a) and for w from (4.1d) yields:

$$\frac{N}{1-N} = \frac{1-\alpha}{\theta}\frac{y}{c} \tag{4.2c}$$

which can be solved for N, since $y/c = (y/k)/(c/k)$. The production function (4.1e) implies

$$\frac{k}{N} = (y/k)^{\frac{1}{\alpha-1}}. \tag{4.2d}$$

Thus, given the solution for N we can back out the stationary values of k, y, c. Given these, equations (4.1a), (4.1c), and (4.1d) can be used to determine λ, r, and w, respectively.

Simultaneously. Instead of solving (4.1) stepwise you may want to input the system in Gauss and use either the Gauss command `EqSolve` or any other program that solves a system of non-linear equations. The Gauss non-linear equations solver needs an initial vector $\mathbf{x}_0 = [k_0, y_0, c_0, i_0, N_0, w_0, r_0, \lambda_0]'$ to start the algorithm. If you do not have prior knowledge of the solution, you can use random numbers distributed on a positive subset of $\mathbb{R}^8$. However, since solving non-linear equations is sometimes a tricky business, there is no guarantee that this approach works in all situations.

Given that you have managed to find the stationary solution of the model, you can proceed to give the program the information from which it will compute the linearized model.

4.2 Linearization

The CoRRAM program has two options that are chosen by the flag `_equations`.

- `_equations=1`: numeric differentiation will be used,
- `_equations=0`: you must provide the coefficient matrices of the log-linearized system.

However, since the quadratic part of the solution always rests on second derivatives of the system (3.2), the option `_equations=0` restricts you to linear approximate solutions. In addition, the option `_GSchur=1` cannot be used. I will provide further details below.

Via Numeric Differentiation. Internally, the program uses two different representations of the linearized model. If the flag `GSchur` is set to 1 (i.e., to true), the program computes

$$B\mathbb{E}_t \begin{bmatrix} \bar{\mathbf{x}}_{t+1} \\ \bar{\mathbf{y}}_{t+1} \end{bmatrix} = A \begin{bmatrix} \bar{\mathbf{x}}_t \\ \bar{\mathbf{y}}_t \end{bmatrix} + C\mathbf{z}_t, \tag{4.3}$$

where the bar denotes (absolute) deviations from the stationary solution, i.e., $\bar{\mathbf{x}}_t \equiv \mathbf{x}_t - \mathbf{x}$ and so forth.[1] The matrices B, A, and C are derived from the derivatives of

$$g^i(\mathbf{x}^1, \mathbf{y}^1, \mathbf{z}^1, \mathbf{x}^2, \mathbf{y}^2, \mathbf{z}^2), \quad i = 1, 2, \ldots, n(x) + n(y) \tag{4.4}$$

with respect to current period variables (superscript 1) and next period variables (superscript 2) evaluated at the stationary solution. If some of the model's equations involve only current period values, the matrix B will be singular. The generalized Schur factorization (see Heer and Maußner (2009b)) is applied to find a linear approximate solution.

If `GSchur=0` the program reduces the dynamic linearized system to a smaller one. Toward this purpose it must know the number of static equations, which you must supply in the scalar `_nu`. First, the program computes the matrices of the system:

$$C_u \bar{\mathbf{u}}_t = C_{x\lambda} \begin{bmatrix} \bar{\mathbf{x}}_t \\ \bar{\boldsymbol{\lambda}}_t \end{bmatrix} + C_z \mathbf{z}_t, \tag{4.5a}$$

$$D_{x\lambda}\mathbb{E}_t \begin{bmatrix} \bar{\mathbf{x}}_{t+1} \\ \bar{\boldsymbol{\lambda}}_{t+1} \end{bmatrix} + F_{x\lambda} \begin{bmatrix} \bar{\mathbf{x}}_t \\ \bar{\boldsymbol{\lambda}}_t \end{bmatrix} = D_u \mathbb{E}_t \bar{\mathbf{u}}_{t+1} + F_u \bar{\mathbf{u}}_t + D_z \mathbb{E}_t \mathbf{z}_{t+1} + F_z \mathbf{z}_t, \tag{4.5b}$$

Second, it reduces this system to

$$\mathbb{E}_t \begin{bmatrix} \bar{\mathbf{x}}_{t+1} \\ \bar{\boldsymbol{\lambda}}_{t+1} \end{bmatrix} = B^{-1}A \begin{bmatrix} \bar{\mathbf{x}}_t \\ \bar{\boldsymbol{\lambda}}_t \end{bmatrix} + B^{-1}C\mathbf{z}_t, \tag{4.6}$$

where

$$\begin{aligned} B &:= \left(D_{x\lambda} - D_u C_u^{-1} C_{x\lambda}\right), \\ A &:= -\left(F_{x\lambda} - F_u C_u^{-1} C_{x\lambda}\right), \\ C &:= \left(D_z + D_u C_u^{-1} C_z\right)\Pi + \left(F_z + F_u C_u^{-1} C_z\right). \end{aligned}$$

This system is solved for the linear part of the solution by applying the Schur factorization to the matrix $B^{-1}A$. Error messages may occur:

- if the matrix C_u is singular,

[1] Since the stationary value of the vector $\mathbf{z}_t$ is the zero vector, $\bar{\mathbf{z}}_t \equiv \mathbf{z}_t$.

- if the matrix B is singular.

The first error message will occur, if the ordering of your variables in the vector $\mathbf{y}_t$ is unfortunate. The program partitions the vector $\mathbf{y}_t$ (see (3.3)) by assigning the first `_nu` variables to the vector $\mathbf{u}_t$ and the remaining $n(y) - n(u)$ variables to the vector $\boldsymbol{\lambda}_t$. Now, suppose that one of your static equations only involves variables from the vectors $\mathbf{x}_t$ and $\boldsymbol{\lambda}_t$. In this case, the respective line of C_u has zero elements and the matrix, thus, is singular. By changing the ordering of the variables, you can work around this problem.

The second error message will occur, if more than $n(u)$ of the linearized equations are static. This will happen, if some of the linearized dynamic equations can be arranged to imply a further static equation. If you are not able to figure this out and change the system accordingly, use `GSchur=1`.

In order to compute the matrices in either (4.3) or (4.5), you must supply the equations in a procedure named `Sys`. This procedure receives the vector

$$\mathbf{w} = [\mathbf{x}^1, \mathbf{y}^1, \mathbf{z}^1, \mathbf{x}^2, \mathbf{y}^2, \mathbf{z}^2]'$$

and returns the right-hand side of (4.4). If `GSchur=0` the program assumes that the first `_nu` equations in `Sys` are static equations.

As an example of `Sys` consider the system (3.4).

```
proc(1)=Sys(w);

  local fx, c1, y1, i1, r1,w1,n1, k1, l1, z1,
            c2, y2, i2, r2,w2,n2, k2, l2, z2;
  k1=w[1];
  y1=w[2];
  c1=w[3];
  i1=w[4];
  r1=w[5];
  w1=w[6];
  n1=w[7];
  l1=w[8];
  z1=exp(w[9]);

  k2=w[10];
  y2=w[11];
  c2=w[12];
  i2=w[13];
  r2=w[14];
  w2=w[15];
  n2=w[16];
```

```
22    l2=w[17];
23    z2=exp(w[18]);
24
25    fx=zeros(_nx+_ny,1);
26
27    fx[1]=(c1^(-eta))*((1-n1)^(theta*(1-eta)))-l1;
28    fx[2]=theta*(c1^(1-eta))*((1-n1)^(theta*(1-eta)-1))-w1*l1;
29    fx[3]=w1-(1-alpha)*z1*(n1^(-alpha))*(k1^alpha);
30    fx[4]=r1-alpha*z1*(n1^(1-alpha))*(k1^(alpha-1));
31    fx[5]=y1-z1*(n1^(1-alpha))*(k1^alpha);
32    fx[6]=y1-c1-i1;
33
34    fx[7]=a*k2-(1-delta)*k1 - i1;
35    fx[8]=beta*(a^(-eta))*l2*(1-delta+r2)-l1;
36
37    if _eqno==0; retp(fx); else; retp(fx[_eqno]); endif;
38    retp(fx);
39
40  endp;
```

In lines 5-23 the variables in the vector `w` are written in scalar variables with obvious notation. This makes it easier to write the equations. Of course, you can use `w[1]`-`w[17]` instead of the auxiliary variables. The code in lines 27-32 evaluates the static functions (3.4a)-(3.4f) and stores the respective results in the vector `fx`. Lines 34 and 35 relate to the dynamic equations (3.4g) and (3.4h). Line 38 returns the vector `fx` to the calling program. The code in line 37 is required to compute second derivatives (which is done equation by equation) and must not be changed!

Linearization by Paper and Pencil. Numeric derivatives cannot be as precise as analytical ones. In addition, by writing the linearized model in terms of percentage instead of absolute deviations, the coefficient matrices usually involve only elasticities so that the numeric differences between their elements are small making matrix factorization less imprecise. Furthermore, it is often not necessary to derive the stationary solutions of all variables.

Consider the benchmark model of (3.4). Its linearized version is an example of:

$$C_u \hat{\mathbf{u}}_t = C_{x\lambda} \begin{bmatrix} \hat{\mathbf{x}}_t \\ \hat{\boldsymbol{\lambda}}_t \end{bmatrix} + C_z \hat{\mathbf{z}}_t, \tag{4.7a}$$

$$D_{x\lambda} \mathbb{E}_t \begin{bmatrix} \hat{\mathbf{x}}_{t+1} \\ \hat{\boldsymbol{\lambda}}_{t+1} \end{bmatrix} + F_{x\lambda} \begin{bmatrix} \hat{\mathbf{x}}_t \\ \hat{\boldsymbol{\lambda}}_t \end{bmatrix} = D_u \mathbb{E}_t \hat{\mathbf{u}}_{t+1} + F_u \hat{\mathbf{u}}_t + D_z \mathbb{E}_t \hat{\mathbf{z}}_{t+1} + F_z \hat{\mathbf{z}}_t, \tag{4.7b}$$

$$\hat{\mathbf{z}}_t = \Pi \hat{\mathbf{z}}_{t-1} + \sigma \Omega \boldsymbol{\nu}_t, \ \boldsymbol{\nu}_t \sim N(\mathbf{0}_{n(z)\times 1}, I_{n(z)}), \tag{4.7c}$$

where the hat denotes relative deviations from the stationary solution, i.e., $\hat{x}_t \equiv (x_t - x)/x$ for any variable x of the model including the shocks. The vectors and matrices are:

$$\hat{\mathbf{u}}_t \equiv [\hat{y}_t, \hat{c}_t, \hat{i}_t, \hat{N}_t, \hat{w}_t, \hat{r}_t]',\ \hat{\mathbf{x}}_t \equiv \hat{k}_t,\ \hat{\boldsymbol{\lambda}}_t \equiv \hat{\lambda}_t, \hat{\mathbf{z}}_t \equiv \hat{Z}_t,$$

$$C_u = \begin{bmatrix} 0 & -\eta & 0 & -\theta(1-\eta)\frac{N}{1-N} & 0 & 0 \\ 0 & (1-\eta) & 0 & -[\theta(1-\eta)-1]\frac{N}{1-N} & -1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 1 \\ -1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & \alpha - 1 & 0 & 0 \\ 1 & -\frac{c}{y} & -\frac{i}{y} & 0 & 0 & 0 \end{bmatrix},$$

$$C_{x\lambda} = \begin{bmatrix} 0 & 1 \\ 0 & 1 \\ -1 & 0 \\ 0 & 0 \\ \alpha & 0 \\ 0 & 0 \end{bmatrix},\ C_z = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, D_{x\lambda} = \begin{bmatrix} a & 0 \\ 0 & 1 \end{bmatrix},\ F_{x\lambda} = \begin{bmatrix} \delta - 1 & 0 \\ 0 & -1 \end{bmatrix},$$

$$D_u = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \beta a^{-\eta}(1-\delta) - 1 \end{bmatrix},$$

$$F_u = \begin{bmatrix} 0 & 0 & a - 1 + \delta & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

$$D_z = F_z = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

If you set `_equations=0`, you must input the matrices `Cu`, `Cxl`, `Cz`, `Dxl`, `Fxl`, `Du`, `Fu`, `Dz` and `Fz`. Also note that in this case the program assumes that the percentage (and not the absolute) deviations of the shock variables are governed by the first-order vector autoregressive process (4.7c).

As in the case of (4.5), the system (4.7) is reduced to the smaller dynamical system

$$\begin{aligned} \mathbb{E}_t \begin{bmatrix} \hat{\mathbf{x}}_{t+1} \\ \hat{\boldsymbol{\lambda}}_{t+1} \end{bmatrix} &= B^{-1} A \begin{bmatrix} \hat{\mathbf{x}}_t \\ \hat{\boldsymbol{\lambda}}_t \end{bmatrix} + B^{-1} C \hat{\mathbf{z}}_t, \\ B &:= \left(D_{x\lambda} - D_u C_u^{-1} C_{x\lambda} \right), \\ A &:= -\left(F_{x\lambda} - F_u C_u^{-1} C_{x\lambda} \right), \\ C &:= \left(D_z + D_u C_u^{-1} C_z \right) \Pi + \left(F_z + F_u C_u^{-1} C_z \right). \end{aligned} \tag{4.8}$$

Thus, you will observe error messages if either C_u or B are not invertible.[2] In this

[2]The solution algorithm does not use matrix inversion but solves the respective linear systems. Yet, it checks whether C_u and B are singular before trying to solve theses systems.

case you must check your derivation of the model for mistakes or your model input for typing errors.

4.3 Structure of Solutions

The approximate solutions of a DSGE model are feed-back rules that determine the model's endogenous variables $\mathbf{x}_{t+1}$ and $\mathbf{y}_t$ as linear or quadratic functions of the model's current states $\mathbf{x}_t$ and realizations of the shocks $\mathbf{z}_t$.

If `_equations=0` the program computes the four matrices that govern the approximate dynamic solution

$$\hat{\mathbf{x}}_{t+1} = L_x^x \hat{\mathbf{x}}_t + L_z^x \hat{\mathbf{z}}_t, \tag{4.9a}$$

$$\hat{\mathbf{y}}_t = L_x^y \hat{\mathbf{x}}_t + L_z^y \hat{\mathbf{z}}_t. \tag{4.9b}$$

If `_equations=1` and `linear=0` the solutions are:

$$x_{it+1} = x_i + (\mathbf{l}_x^i)'\bar{\mathbf{x}}_t + (\mathbf{l}_z^i)'\bar{\mathbf{z}}_t + \frac{1}{2}\begin{bmatrix}\bar{\mathbf{x}}_t', & \mathbf{z}_t', & \sigma\end{bmatrix} H^i \begin{bmatrix}\bar{\mathbf{x}}_t \\ \mathbf{z}_t \\ \sigma\end{bmatrix}, \ i = 1, \ldots, n(x), \tag{4.10a}$$

$$y_{jt} = y_j + (\mathbf{l}_x^j)'\bar{\mathbf{x}}_t + (\mathbf{l}_z^j)'\bar{\mathbf{z}}_t + \frac{1}{2}\begin{bmatrix}\bar{\mathbf{x}}_t', & \mathbf{z}_t', & \sigma\end{bmatrix} H^j \begin{bmatrix}\bar{\mathbf{x}}_t \\ \mathbf{z}_t \\ \sigma\end{bmatrix}, \ j = 1, \ldots, n(y). \tag{4.10b}$$

If `_linear=1` the program returns only the linear part of this solution.

4.4 Additional Variables

Occasionally you may want solutions for variables that cannot be included in $\mathbf{y}_t$, since they depend on the model's solution. For instance, in the benchmark model the current price p_t of a bond that pays one unit of consumption for certain in the next period is given by

$$p_t = \beta a^{-\eta} \mathbb{E}_t \frac{\lambda_{t+1}}{\lambda_t},$$

so that

$$\hat{p}_t = \mathbb{E}_t \hat{\lambda}_{t+1} - \hat{\lambda}_t.$$

From the linear solution (where $\mathbf{l}_x^\lambda$ denotes the row of L_x^y that refers to the coefficients of the solution for λ)

$$\hat{\lambda}_t = (\mathbf{l}_x^\lambda)'\hat{x}_t + (\mathbf{l}_z^\lambda)'\hat{z}_t$$

you get

$$\begin{aligned}\mathbb{E}_t\hat{\lambda}_{t+1} &= \mathbb{E}_t\left\{(\mathbf{l}_x^\lambda)'\hat{x}_{t+1} + (\mathbf{l}_z^\lambda)'\hat{z}_{t+1}\right\},\\ &= (\mathbf{l}_x^\lambda)'\left\{L_x^x\hat{x}_t + L_z^x\hat{z}_t\right\} + (\mathbf{l}_z^\lambda)'\mathbb{E}_t\left\{\Pi\hat{z}_t + \epsilon_{t+1}\right\},\\ &= (\mathbf{l}_x^\lambda)'L_x^x\hat{x}_t + \left[((\mathbf{l}_x^\lambda)'L_z^x + (\mathbf{l}_z^\lambda)'\Pi\right]\hat{z}_t.\end{aligned}$$

Therefore, you can determine $\hat{p}_t$ as a linear function of $\hat{x}_t$ and $\hat{z}_t$ with matrices:[3]

$$\begin{aligned}L_x^s &= (\mathbf{l}_x^\lambda)'(L_x^x - I_{n(x)}),\\ L_z^s &= (\mathbf{l}_x^\lambda)'L_z^x + (\mathbf{l}_z^\lambda)'(\Pi - I_{n(z)}).\end{aligned} \tag{4.11}$$

Note that the program must compute the linear solution before it can determine L_x^s and L_z^s. For this reason you must input the two matrices before your simulate the model. Also, you must set the scalar `_ns` to the number of additional variables and position these variables in the vector $\mathbf{y}$ to the right of the model's original variables.

Note also that this opportunity is available to you in the case `_equations=0` only!

4.5 Difference Stationary Growth

The computation of impulse responses and second moments depends on the interpretation of the model's variables. In the model of equations (3.4) the variables are scaled by the level of labor augmenting technical progress A_t which growth deterministically at the rate $a-1$. Many DSGE models, however, assume that A_t grows according to the law

$$\begin{aligned}a_t &= \frac{A_{t+1}}{A_t},\\ a_t &= e^{\ln a + \ln z_t},\\ \ln z_t &= \rho\ln z_{t-1} + \sigma\epsilon_t, \quad \epsilon_t \sim N(0,1),\end{aligned} \tag{4.12}$$

so that $\ln A_t$ is a difference stationary growth process. In this case, the program assumes that you have defined stationary variables according to $X_t = x_t A_{t-1}^\xi$, where X_t is the level of a variable, x_t is the solution for the scaled variable computed by the program, and the exponent ξ is either zero (if the variable needs not to be scaled, as for instance hours N_t), equal to $\xi = 1$ (if division by A_{t-1} makes a variable stationary), or equal to $\xi = \eta$, as in the case of the Lagrange multiplier in the benchmark business cycle model, where η is coefficient of relative risk aversion.

[3]The program uses the matrices `Lsx` and `Lsz` to denote the relation between the additional variables in the subvector `s` and the variables in $\hat{\mathbf{x}}_t$ and $\hat{\mathbf{z}}_t$. In this one variable example, L_x^s and L_z^s are row vectors.

To get the correct interpretation of impulse responses and second moments[4] you must

- set the flag `_DS=1`,
- provide information on ξ in the structure `_Var` (see below),
- use the variable a_t as the first element in the vector $\mathbf{y}_t$.

4.6 Error Messages

DSGE models have determinate solutions if:

- $n(x)$ of the eigenvalues of $B^{-1}A$ or $n(x)$ of the eigenvalues of the matrix pencil (B, A) are inside the unit circle, and
- $n(\lambda)$ of the eigenvalues $B^{-1}A$ or $n(y)$ of the eigenvalues of the matrix pencil (B, A) are outside of the unit circle.

The program stops with an error message if these two conditions are not satisfied or if it is not able to factorize either $B^{-1}A$ or (B, A). The latter usually points to errors in your code.

When the program computes the linear part of the solution it uses complex arithmetic. If the matrices of your model are large and/or rather unbalanced, roundoff errors may cause the complex part of the solution matrices to be non-negligible. If this happens, the program issues a warning message. Please check if errors in your code cause this problem or if changing the matrix factorization method (see above, the setting of `GSchur`) solves the problem.

5 Example Program

This section explains the code in the file `CoRRAM_Test_2.g` which solves and simulates the model of (3.4) using the model's equations.

```
1  @ ----------- CoRRAM_Test_3.g -------------------------
2
3      Alfred Maußner
4      17 September 2012
5
6      Purpose: Solve the benchmark business cycle model
```

[4]See Appendix C of Heer and Maußner (2010) for the computation of impulse responses in such a model.

```
7               of Heer and Maußner (2009) using the
8               CoRRAM toolbox.
9
10 ------------------------------------------------------- @
11
12 // Do not change the next there lines of code:
13 new;
14 library user, pgraph, corram;
15 #include CoRRAM.sdf;
16
17 nx=1;
18 ny=7;
19 nu=6;
20 nz=1;
21 ns=0;
22 _PQG=1;
23 _loadShocks=1;
24
25 NewModel("Benchmark-1.txt",nx,ny,nu,nz,ns);
26
27 _Var[1].name="Output";
28 _Var[1].type="y";
29 _Var[1].pos=1;
30 _Var[1].print=1;
31 _Var[1].crosscorr=1;
32 _Var[1].plot=1;
33 _Var[1].plotno=2;
34 _Var[1].relsx=1;
35
36 _Var[2].name="Consumption";
37 _Var[2].type="y";
38 _Var[2].pos=2;
39 _Var[2].print=1;
40 _Var[2].crosscorr=0;
41 _Var[2].plot=1;
42 _Var[2].plotno=2;
43
44 _Var[3].name="Investment";
45 _Var[3].type="y";
46 _Var[3].pos=3;
47 _Var[3].print=1;
48 _Var[3].crosscorr=0;
49 _Var[3].plot=1;
50 _Var[3].plotno=2;
51
```

```
_Var[4].name="Hours";
_Var[4].type="y";
_Var[4].pos=4;
_Var[4].print=1;
_Var[4].crosscorr=0;
_Var[4].plot=1;
_Var[4].plotno=2;

_Var[5].name="Real Wage";
_Var[5].type="y";
_Var[5].pos=5;
_Var[5].print=1;
_Var[5].crosscorr=1;
_Var[5].plot=1;
_Var[5].plotno=3;

_Var[6].name="User Costs of Capital";
_Var[6].type="y";
_Var[6].pos=6;
_Var[6].print=1;
_Var[6].crosscorr=0;
_Var[6].plot=1;
_Var[6].plotno=3;

_Var[7].name="Marginal Utility";
_Var[7].type="y";
_Var[7].pos=7;
_Var[7].print=1;
_Var[7].crosscorr=0;
_Var[7].plot=1;
_Var[7].plotno=3;

_Var[8].name="Capital Stock";
_Var[8].type="x";
_Var[8].pos=1;
_Var[8].print=0;
_Var[8].crosscorr=0;
_Var[8].plot=1;
_Var[8].plotno=4;

_Var[9].name="TFP Shock";
_Var[9].type="z";
_Var[9].pos=1;
_Var[9].print=0;
_Var[9].crosscorr=0;
```

```
_Var[9].plot=1;
_Var[9].plotno=1;

/* Set legend positions individually */
_plegend[1,1,1]=10;
_plegend[1,1,2]=0.5;

_plegend[1,2,1]=10;
_plegend[1,2,2]=3.5;

_plegend[1,4,1]=6;
_plegend[1,4,2]=0.5;

// The parameter of the model
a=1.005;
alpha=0.27;
beta=0.994;
eta=2.0;
delta=0.011;
rhoZ=0.90;
sigmaz=0.0072;
nstar=0.13;

_Rho[1,1]=RhoZ;
_Omega[1,1]=SigmaZ;
_Sigma=1;

// Compute the stationary solution
yk=(a^eta-beta*(1-delta))/(beta*alpha);
ck=yk+(1-a-delta);
theta = (1-alpha)*(yk/ck)*(1-nstar)*(1/nstar);
kn=yk^(1/(alpha-1));
kstar=kn*nstar;
ystar=yk*kstar;
cstar=ck*kstar;
istar=ystar-cstar;
wstar=(1-alpha)*(ystar/nstar);
rstar=alpha*yk;
lstar=(cstar^(-eta))*((1-nstar)^(theta*(1-eta)));

// assign the stationary values
_var[1].star=ystar;
_var[2].star=cstar;
_var[3].star=istar;
_var[4].star=nstar;
```

```
142  _var[5].star=wstar;
143  _var[6].star=rstar;
144  _var[7].star=lstar;
145  _var[8].star=kstar;
146  _var[9].star=0;
147
148  // solve the model
149  retc=SolveModel;
150
151  // simulate the model
152  if retc;
153      SimulateModel(25,80,500);
154  endif;
155
156  ende:
157
158  end;
159
```

The file header in lines 1-10 provides information on the author of the program, the date at which the program was written, and the purpose of the program. Lines 13-15 must be present in each program. The `new` command clears the Gauss memory. The command in line 14 loads the Gauss libraries required by the program. These provide information to Gauss where the CoRRAM procedures are stored. The command in line 15 loads the definitions of several Gauss structures that are used by CoRRAM.

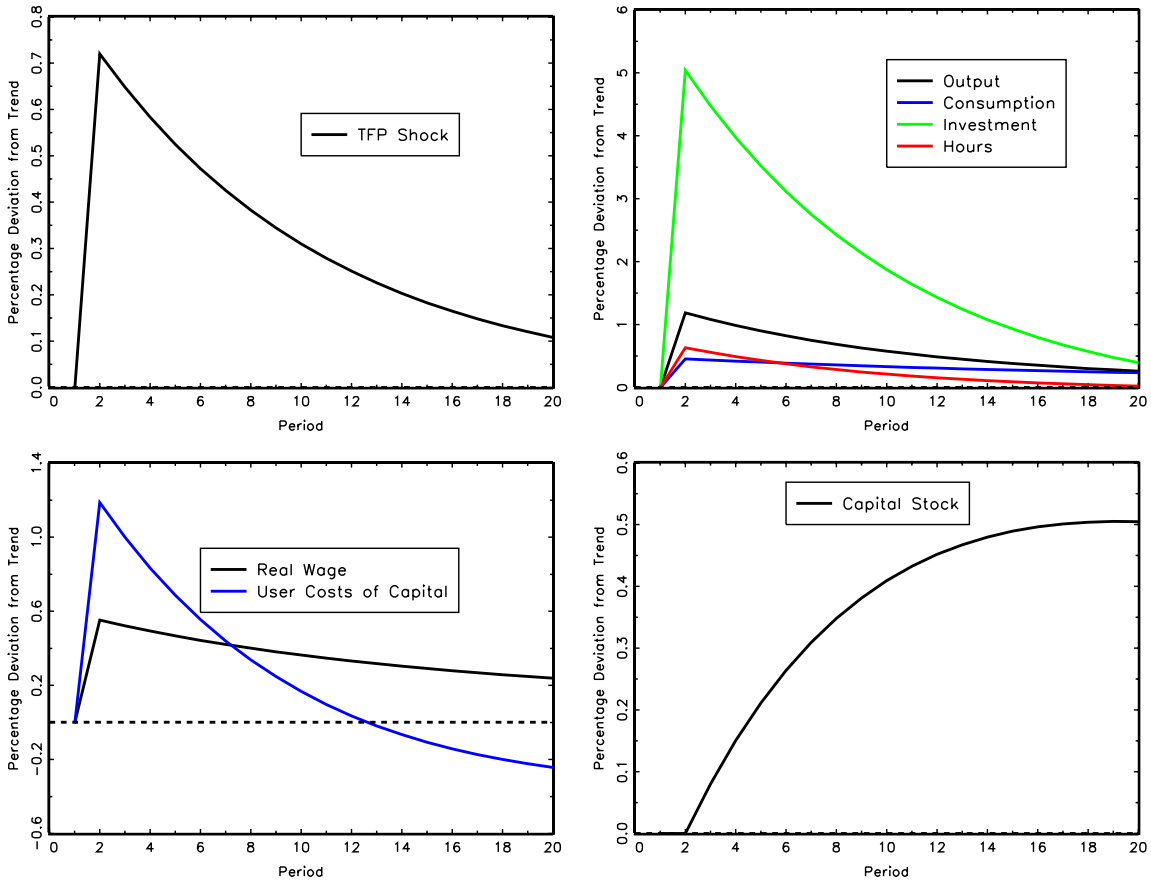
Lines 17-21 provide information on the number of variables. In lines 22 and 24 two of the default options are changed. The option `_LoadShocks=1` instructs the program to load the random numbers used to simulate the model from a file. Depending on the flag `_GLight` this file has the extension `.fmt` (`_Glight=0`) or `.xls` (`_Glight=1`). You will get an error message if this file is not present in the current working directory. You can create this file with the command `MakeShocks(nz,nobs,nofs)` (place it before the command `SimulateModel` or issue this command from the command prompt.) `nz`, `nobs`, and `nofs` refer to the number of shock, the length of the simulated time series, and the number of simulations, respectively.

The command in line 25 sets up a new model. Output (including error messages) will be directed to the file `Benchmark-1.txt` in the current working directory.

The code in lines 27-98 provides CoRRAM with information on the variables (names, types,positions,stationary values,scaling,print and output options) which I will explain below in more detail.

In lines 104-108 the legends in the Gauss publication quality graphics are repositioned (see below). The statements in lines 111-118 assign values to the parameters of

Figure 5.1
Impulse Responses from the Benchmark Business Cycle Model



the model. The code in lines 120-122 provides CoRRAM with information about the matrices Π (`_Rho` in CoRRAM) and Ω.

Lines 125-146 compute and assign the stationary values of the variables. The statement in line 149 solves the model, i.e., it computes either the linear or quadratic approximate solution. The procedure `SolveModel` returns `retc=1` if the model could be solved and zero otherwise.

The statement in line 153 computes impulse responses and second moments. The number of periods for which impulse responses are computed is given in `nobs1`. If this is set equal to zero, the program does not compute and plot impulse responses. Depending on the flag `_PQG`, the program will plot the impulse responses either with the Gauss publication quality graphic routines (`_PQG=1`) or with the new graphic routines implemented since Gauss version 12.1. For instance, the program's graphic output for the benchmark business cycle model may look like Figure 5.1

There are two other options that determine the way impulse responses are plotted. If `_scale=0`, each graphics window will be scaled individually. This option does not

work under `_PQG=0`. For `_scale=1` the same scale will be applied to all panels. Impulse responses will be colored if `_color=1`, otherwise black lines will be used.

CoRRAM computes second moments from `nofs` simulations of length `nobs2` each. The program will not compute second moments if `nobs2=0`. If the flag `_confidence` is set equal to one, the program computes and prints 95 percent intervals for the simulated moments. If you run Gauss Light, and have set the flag `_GLight=1`, the program will ignore this option, since it requires more storage than Gauss Light allows. Table 5.1 presents example output for the model in (3.4).

Table 5.1
Sample output of `CoRRAM`

```
Second moments from simulated data:

Output                  1.44    1.00    1.00    0.65
Consumption             0.56    0.39    0.99    0.66
Investment              6.13    4.26    1.00    0.64
Hours                   0.77    0.54    1.00    0.64
Real Wage               0.67    0.47    0.99    0.66
Rental Rate of Capital  1.48    1.03    0.98    0.64

Column 1: Variable name
Column 2: Standard deviation
Column 3: Standard deviation relative to standard deviation of Output
Column 4: Cross correlation with Output
Column 5: first order autocorrelation
```

Finally, lines 160-201 (not shown) define the procedure `Sys` as explained in Section 4.2.

6 Program Options

The Gauss structure `_Var` provides the program with information about the model's variables and controls the program's output. There must be as many instances of this structure as there are variables in your model. The procedure `NewModel` initializes this structure. It is then to up to the user to assign the appropriate values to the structure. I explain this with an example.

```
1  _Var[1].name="Output";
2  _Var[1].type="y";
3  _Var[1].pos=1;
4  _Var[1].print=1;
5  _Var[1].relsx=1;
6  _Var[1].crosscorr=1;
7  _Var[1].plot=1;
8  _Var[1].plotno=1;
9  _Var[1].star=ystar;
10 _Var[1].xi=1;
```

`_Var[1]` refers to the first instance of the structure `_Var`. This instance stores information on the variable named `Output`. This variable is a member of the vector $\mathbf{y}_t$. The structure member `type` can be assigned the strings `x`, `y`, and `z`, depending on to which vector of variables it belongs. In the example, `output` is the first variable in the vector `y` (see line 3). In lines 4-7, the assignment of 1 stands for true and 0 for false. Thus, `print=1` tells the program that you want information on the second moments of the variable `output` to be printed. Without any further information the program prints the standard deviation of this variable and its first-order autocorrelation. If you also set `relsx` and `crosscorr` to true, the output table has two additional columns that show the standard deviations of other variables relative to the standard deviation of `output` and the cross-correlations of other variables with output.

In line 7 and 8 you tell the program to print the impulse response of `output` to the graphics panel 1. Accepted values for `plotno` are the integers 1 through 6. Finally, you may want to overwrite the program's automatic positioning of plot legends. In this case, you must edit the global variable `_plegend`. If you want to change the lower left corner of the legend in panel i of figure j to the plot coordinates (x, y), type

```
1  _plegend[j,i,1]=x;
2  _plegend[j,i,2]=y;
```

Note, this works only if `_PQG=1`. For `_PQG=0` you must position the legend box manually in the Gauss graphics tab.

7 Summary of Global Control Variables

Table 7.1 explains the settings of the variables that determine the solution method and the output options of CoRRAM.

Table 7.1
Global Control Variables of CoRRAM

Variable	Possible Settings and Effect		Default
_color	1	colored impulse responses are plotted	1
	0	all impulse responses are black.	
_confidence	1	the program prints 95 percent intervals for the computed second moments	
	0	the program prints only averages of the second moments	0
_DS	1	the program assumes a difference stationary growth process. The variable a_t must be the first element in the vector $\mathbf{y}_t$.	
	0	the program assumes that the variables in your model are stationary, either since there is no growth or since the growth process is deterministic.	0
_equations	1	the program expects the equations of your model in Sys	1
	0	you must input the coefficient matrices of the log-linearized model	
_GaussOnly	1	no foreign language code will be loaded, the generalized Schur factorization is then not available.	
	0	Fortran routines from Tools.dll will be used.	0
_Glight	1	if _LoadShocks=1 random numbers will be read from an Excel spreadsheet which you can create with the procedure MakeShocks.	
	0	if _LoadShcoks=1 random numbers will be read from a Gauss matrix file. In addition, you may use the flag _confidenc=1.	0
_GSchur	1	the generalized Schur factorization will be used	
	0	the Schur factorization will be used	0
_HPL		the filter weight of the HP-filter	1600
	0	the HP-filter will not be applied	
_plegend		three dimensional array that stores information on the individual positioning of legends.	
_linear	1	the program computes only linear feed back rules	
	0	the program computes quadratic feed back rules	0
_LoadShocks	1	the program uses random numbers from the file eps_array.fmt or eps_array.xls	
	0	the program generates new random numbers for each simulation.	0
_PQG	1	impulse responses will be plotted using publication quality graphic commands.	1
	0	impulse responses will be plotted using the new (since Gauss version 12.1) plotXY command.	
_QA_type	1	second order solution is based on the chain rule for Hessian matrices	1
	0	second order solution is based on tensor products	
_scale	1	all graphic panels share the same scale	
	0	all graphic panels will be scaled individually.	1
_Sylvester	1	if _QA_type=1 the routine sylvester in Tools.dll is used	1
	0	if _QA_type=1 the vec operator is used	

Besides the control variables there are a number of other global variables used by the program. All global variables begin with an underscore _. Please do not use variable names that begin with an underscore. This avoids that you interfere with the program code. Table 7.2 displays variables used by CoRRAM. Please adhere to these naming conventions.

Table 7.2
Symbol Table

Symbol Name	Equivalent Symbol or Function
_Cu	C_u from (4.7a)
_Cxl	$C_{x\lambda}$ from (4.7a)
_Cz	C_z from (4.7a)
_Dxl	$D_{x\lambda}$ from (4.7b)
_Fxl	$F_{x\lambda}$ from (4.7b)
_Du	D_u from (4.7b)
_Fu	F_u from (4.7b)
_Dz	D_z from (4.7b)
_Fz	F_z from (4.7b)
_sigma	σ from (3.1b) or (4.7c)
_Omega	Ω from (3.1b) or (4.7c)
_Rho	Π from (3.1b) or (4.7c)
_nx	$n(x)$ the number of variables in the vector $\mathbf{x}_t$
_ny	$n(y)$ the number of variables in the vector $\mathbf{y}_t$
_nz	$n(z)$ the number of variables in the vector $\mathbf{z}_t$
_ns	the number of additional variables (see Section 4.4)
_nu	the number of static equations
_xstar	the vector with the stationary solutions of the state variables
_ystar	the vector with the stationary solutions of the non predetermined variables
_wstar	the vector _xstar|_ystar|_zstar
_xt	three dimensional array that stores the impulse responses of the variables in $\mathbf{x}_t$.
_yt	three dimensional array that stores the impulse responses of the variables in $\mathbf{y}_t$.
_zt	three dimensional array that stores the impulses from $\mathbf{z}_t$.
Sys	the procedure with the model's equations from (3.2)
_Lxx	L_x^x from (4.9)
_Lxz	L_z^x from (4.9)
_Lyx	L_x^y from (4.9)
_Lyz	L_z^y from (4.9)
_Lsx	L_x^s from (4.11)
_Lsz	L_z^s from (4.11)
_xcube	three dimensional array, page j stores H^j from (4.10a)
_ycube	three dimensional array, page j stores H^j from (4.10b)

8 Inside the Black Box

If the standard options provided to you via the command NewModel, SolveModel, and SimulateModel satisfy your needs, you can skip this section and start right away with setting up your own model or run CoRRAM_Test_1.g and CoRRAM_Test_2.g. Otherwise, the information in this section will help you to use the main procedures that make up CoRRAM.

8.1 Linear Solutions

If you do not input the matrices from the system (4.7), the code in the procedure `SolveModel` in the file `CoRRAM_1.src` sets up the matrices from which the linear part of the solution will be computed. If `_GSchur=0` (the default), the procedure `SolveReducedModel` builds the model in (4.6) and solves it with a call to `SolveLA1` (in `CoRRAM_4.src`), otherwise `SolveModel` builds the model in (4.3) and solves it with a call to `SolveLA2` (also in the file `CoRRAM_4.src`). Depending on the flag `_GaussOnly`, the procedure `SolveLA1` either uses Gauss commands to compute the Schur factorization of the matrix $B^{-1}A$ or calls a procedure in the dynamic link library `Tools.dll` to get the factorization. The option `GSchur=1` always calls a procedure in `Tools.dll` since Gauss provides no command to compute the generalized Schur factorization. When `SolveModel` is finished, the matrices `_Lxx`, `_Lxz`, `_Lyx`, and `_Lyz` store the solution matrices from (4.9).

8.2 Quadratic Part of the Solution

Given the linear part of the solution, either the program `SolveQA2` or the program `SolveQA3` (both in the file `CoRRAM_5.src`) computes two three dimensional arrays that store the matrices H^i and H^j from (4.10). The respective call is:

```
{_xcube,_ycube}=SolveQA2(_Lxx,_Lxz,_Lyx,_Lyz,_gmat,hcube);
```

or

```
{_xcube,_ycube}=SolveQA3(_Lxx,_Lxz,_Lyx,_Lyz,_gmat,hcube);
```

If the global control variable `_QA_type` is true (i.e. set equal to one, the default setting) `SolveQA2` is used.

The matrix `_gmat` stores the Jacobian matrix of the system (4.4), and the three dimensional array `hcube` holds the Hesse matrices of the individual equations $g^i(\cdot)$ of (4.4). They are computed by `CDJac` and `CDHesse`, which employ central difference formulas to approximate first and second derivatives, respectively. Each page of `_xcube` stores one of the $n(x)$ matrices H^i of (4.10a). Analogously, the $n(y)$ pages of `_ycube` store the matrices H^j of (4.10b). The procedure `PF` uses `_Lxx`, `_Lxz`, `_Lyx`, `_Lyz`, `_xcube`, and `_ycube` and returns x^i_{t+1} and y^j_t as functions of $\bar{\mathbf{x}}_t$ and $\bar{\mathbf{z}}_t$, so that you can use this procedure to simulate your model. The respective call is:

```
xi = PF("x",i,whut),
yj = PF("y",j,whut),
```

where `i` and `j` refer to the position number (starting from the left) of x_i and y_j in the vectors $\mathbf{x}_t$ and $\mathbf{y}_t$, respectively. The vector `whut` stores $\mathbf{w}_t := [\bar{\mathbf{x}}'_t, \bar{\mathbf{z}}'_t, \sigma]'$.

8.3 Simulation

The procedures `Impulse1` and `Impulse2` compute the impulse responses. `Impulse1` is invoked, if `_equations=0`. Both procedures return the percentage deviations of the model's variables in the matrices `xt`, `yt`, and `zt`. Each row of these matrices stores `nobs1` elements of the response of the respective variable to a one-standard deviation shock $\nu_{l2} = \sigma\Omega\mathbf{e}_l$, where $\mathbf{e}_l$ is the row vector with its l-th element equal to one and zeros elsewhere. The impulse response of the state variables are stored in `xt`, those of the non-predetermined variables in `yt`, and those of the shocks in `zt`. The program computes as many impulse responses as are elements in the vector $\mathbf{z}_t$. They are stored in three arrays, `_xt`, `_yt`, and `_zt`, respectively. Page i refers to shock $i \in \{1, 2, \ldots, n(z)\}$. The order of the variables in the columns of each page confirms to the numbering of the variables specified in the structure `_Var`. When the program is finished, these arrays stay in the Gauss memory until a new version of the program is run or the Gauss window is closed.

Analogously, the procedures `RBCRun1` and `RBCRun2` simulate the model for ongoing shocks. If `_confidence=0` they return a vector `sx` and a matrix `rx`. The vector `sx` stores the standard deviations (in percent) of the variables. The first $n(x)$ elements refer to the state variables, the second $n(y)$ elements refer to the non-predetermined variables and third $n(z)$ elements relate to model's shocks. The matrix `sx` has dimension $2n \times 2n$, $n = n(x) + n(y) + n(z)$. Its first $n \times n$ block stores the contemporaneous correlation between the elements of the vector

$$\mathbf{v}_t = \left[x_{1t}, \ldots, x_{n(x)t}, y_{1t}, \ldots, y_{n(y)t}, z_{1t}, \ldots, z_{n(z)t}\right]'.$$

The block with elements sx_{ij}, $i = 1, \ldots, n$, $j = n+1, \ldots, 2n$, holds the cross-correlations between the elements of $\mathbf{v}_t$ and $\mathbf{v}_{t-1}$.

If `_confidence=1`, `sx` is a matrix. Column i of this matrix stores the standard deviations of the elements of the vector $[\mathbf{x}, \mathbf{y}, \mathbf{z}]'$ from the i-th simulation of the model. Analogously, `rx` is a three dimensional array, where the i-th page stores the contemporaneous and one-period lagged correlations between the model's variables.

9 References

Gomme, Paul and Paul Klein. 2011. Second-Order Approximation of Dynamic Models Without the Use of Tensors. *Journal of Economic Dynamics & Control*. Vol. 35. 604-615.

Hansen, Gary D. 1985. Indivisible Labor and the Business Cycle, *Journal of Monetary Economics*. Vol. 16. 309-327.

Heer, Burkhard und Alfred Maußner. 2009a. Dynamic General Equilibrium Modeling. Springer: Berlin

Heer, Burkhard und Alfred Maußner. 2009b. Computation of Business-Cycle Models with the Generalized Schur Method. *Indian Growth and Development Review*. Vol. 2. 173-182

Heer, Burkhard and Alfred Maußner. 2010. Inflation and Output Dynamics in a Model with Labor Market Search and Capital Accumulation. *Review of Economic Dynamics*. Vol. 13. pp. 654-686.